Taller Guiado - Análisis de Algoritmos

Alberto Luis Vigna Arroyo, Camilo José Martinez

2 de febrero de 2024

Abstract

En este documento se presenta el análisis del algoritmo de ordenamiento Bubble Sort, Insertion Sort y Quick Sort. Cada uno de estos algoritmos aborda el problema de ordenar una lista de elementos, pero difieren en sus enfoques y eficiencias. A continuación, se proporciona un resumen de los aspectos más destacados de cada algoritmo, destacando sus características, ventajas y desventajas, así como sus complejidades temporales y espaciales. Este análisis permite comprender mejor la idoneidad y el rendimiento de cada algoritmo en diferentes contextos y escenarios.

Part I
Análisis y diseño del problema

1 Análisis

El problema, informalmente, se puede describir como calcular para una secuencia de elementos $S = \langle a_1, a_2, a_3, \cdots, a_n \rangle$ en donde $\forall_i a_i \in \mathbb{T}$ y $\mathbb{T}$ existe una relación de orden parcial $' \leq'$, una secuencia ordenada, es decir una permutación $S = \langle a'_1, a'_2, a'_3, \cdots, a'_n \rangle$ en donde $a_i \leq a'_{i+1}$ y $a'_i \in S$.

2 Diseño

Con las observaciones presentadas en el análisis anterior, podemos escribir el diseño de un algoritmo que solucione el problema . A veces este diseño se conoce como el «contrato» del algoritmmos o las «precondiciones» y «poscondiciones» del algoritmo. El diseño se compone de entradas y salidas:

Definition. Entradas:

1. Definición entrada 1
2. Definición entrada 2

Definición. Salidas:

1. Definición salida 1
2. Definición salida 2

Parte II
Algoritmos

3 Opción algoritmo 1 - BubbleSort

3.1 Algoritmo

Este algoritmo se basa en el principio de comparar pares de elementos adyacentes en una lista y realizar intercambios si es necesario, repitiendo este proceso hasta que la lista esté completamente ordenada. En cada iteración se va recorriendo la lista y verificando si el elementos $a_i > a_{i+1}$. Si este lo es los elementos se "rotan" para que estos vayan siendo ordenados.

Algorithm 1 BubbleSort

```
1:  procedure BubbleSort(S)
2:      n ← |S|
3:      for i ← 1 to n do
4:          for j ← 1 to n − i do
5:              if S[j] > S[j + 1] then
6:                  variableTemporal = S[j]
7:                  S[j] = S[j + 1]
8:                  S[j + 1] = variableTemporal
9:              end if
10:         end for
11:     end for
12: end procedure
```

3.2 Complejidad

El algoritmo Bubble Sort tiene órden de complejidad $O(n^2)$. Esto se debe a que emplea dos bucles anidados, donde el bucle externo realiza n iteraciones, y el bucle interno realiza comparaciones y posibles intercambios adyacentes. Esta estructura cuadrática resulta en un rendimiento menos eficiente, especialmente para conjuntos de datos extensos.

3.3 Invariante - Una propiedad que se mantiene verdadera durante la ejecución del algoritmo, para corroborar su funcionamiento

- **Inicio**: Antes de iniciar el ordenamiento consideramos que la parte derecha de la lista (desde el índice $n - i$ hasta $n - 1$) contiene los elementos más grandes y ya está ordenada. Inicialmente, $i = 0$, por lo que la lista completa se toma como no ordenada.

- **Avance**: En cada iteración del bucle externo, el bucle interno compara y posiblemente intercambia elementos adyacentes. Esta operación mueve el elemento más grande a la posición $n-i-1$, manteniendo la parte derecha de la lista ordenada. El valor de i se incrementa después de cada iteración del bucle externo. Si no hay intercambios durante una iteración completa del bucle externo, significa que la lista está completamente ordenada y el algoritmo se detiene.

- **Terminación:** El proceso continua hasta que $i = n - 1$, momentos en el cual se ha completado el ordenamiento de la lista.

3.4 Notas de implementación

4 Opción algoritmo 2 - Insertion Sort

4.1 Algoritmo

Este algoritmo se basa en el principio de iterar sobre una lista S de n elementos, donde $n = |S|$. La idea es que este algorimo empiece desde el segundo elemento ($i = 1$) y que en cada iteración se compare $S[i]$ con los elementos anteriores (es decir, $S[i-1], S[i-2],$

lo compare con los elementos anteriores (los que se encuentren a su derecha), desplazandolos a su derecha mientras que $S[i]$ sea menor.

Parte III
Comparación de los algoritmos